Apple Assembly Line

$1.50

Volume 3 -- Issue 5 February, 1983

In This Issue...

S-C Macro Assembler ///

The Apple /// version of the S-C Macro Assembler is coming right along! I am now selling a preliminary "as is" version for $100. That buys you the assembler, a few pages of documentation about the differences from the Apple][version, and free updates until the finished product appears. This is a working assembler for producing free-running programs; it assembles itself just fine. The biggest gap is the ability to produce relocatable modules for Pascal or BASIC. That will be added next. Call or write if you are interested in being among the first to have this new enhancement to the Apple ///.

Zero-Insertion-Force Game Socket Extender

One of the first things I did to my Apple back in 1977 was to plug a ZIF socket into the game connector. Not too easy, because it first has to be soldered to a header, but I did it.

Now I have discovered a source for a ready-made device that does the same thing, plus brings the socket outside the Apple (if you so desire). There's a picture of the device on page 14. For only $20 I'll send you one!

Really Adding ASCII Strings...............Bob Sander-Cederlof

Last month I promised a "reasonably useful" program to add two numbers together from ASCII strings. I promised:

- Callable from Applesoft, using &.
- Automatic passing of string parameters.
- Allow operands of unequal length.
- Automatic alignment of decimal points.
- Allow negative numbers.
- Handle sums longer than operands.
- Allow leading blanks on operands.
- Allow operands and results up to 253 bytes long!

Okay! It took me three days, but I did it! Of course, the program has grown from 12 lines and 26 bytes of code to over 290 lines and over 450 bytes, too.

The program is now assembled to load at $9000, but you can choose other positions by changing line 1130. I set HIMEM:36864 before doing anything else in the Applesoft program, and then BRUN B.STRING ADDER.

When B.STRING ADDER is BRUN, only the setup code in lines 1160-1220 is executed. What this does is link in the ampersand (&) to the body of my program. Once the "&" is linked, my program responds to a call like "& +$,A$,B$,C$" by adding the numeric values represented in ASCII in A$ and B$ and storing the sum as a string in C$.

When an &-line occurs, Applesoft branches to my line 1520. Lines 1520-1600 check for the characters "+$," after the ampersand. If you don't like those characters, change them to something else. Anyway, if the characters do not match, you get SYNTAX ERROR. If they do match, it is time to collect the three strings variables.

Lines 1620-1690 collect the three string variables. The first two are the operands, the third is the result string. I save the address and length of the actual data of the operand strings. All I save at this point for the result string is the address of the variable descriptor. I call the subroutine PARSE.STRING.NAME to check for a leading comma, search for the variable name, and store the length and address of the referenced string data.

Lines 1730-1860 scan each operand string in turn to find the decimal point position. The routine SCAN divides a string at the decimal point (or where the decimal point would be if there was one), and returns in Y the number of characters to the left of the decimal point. SCAN returns in X the count of the

S-C Macro Assembler (the best there is!)		$80.00
Upgrade from Version 4.0 to MACRO		$27.50
Source code of Version 4.0 on disk		$95.00
Fully commented, easy to understand and modify to your own tastes.		
S-C Macro Assembler ///		$100.00
Preliminary version. Call or write for details.		
Applesoft Source Code on Disk		$50.00
Very heavily commented. Requires Applesoft and S-C Assembler.		
ES-CAPE: Extended S-C Applesoft Program Editor		$60.00
AAL Quarterly Disks		each $15.00
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.		
QD#1: Oct-Dec 1980 QD#2: Jan-Mar 1981 QD#3: Apr-Jun 1981		
QD#4: Jul-Sep 1981 QD#5: Oct-Dec 1981 QD#6: Jan-Mar 1982		
QD#7: Apr-Jun 1982 QD#8: Jul-Sep 1982 QD#9: Oct-Dec 1982		
Double Precision Floating Point for Applesoft		$50.00
Provides 21-digit precision for Applesoft programs. Includes sample Applesoft subroutines for standard math functions.		
FLASH! Integer BASIC Compiler (Laumer Research)		$79.00
Source Code for FLASH! Runtime Package		$39.00
Super Disk Copy III (Sensible Software)	(reg. $30.00)	$27.00
Amper-Magic (Anthro-Digital)	(reg. $75.00)	$67.50
Amper-Magic Volume 2 (Anthro-Digital)	(reg. $35.00)	$30.00
Quick-Trace (Anthro-Digital)	(reg. $50.00)	$45.00
Cross-Reference and Dis-Assembler (Rak-Ware)		$45.00
The Incredible JACK!		$79.00
Blank Diskettes (with hub rings)		package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks		$36.00
Vinyl disk pages, 6"x8.5", hold one disk each		10 for $6.00
Reload your own NEC PC-8023 ribbon cartridges		each ribbon $5.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges		each $2.50
Diskette Mailing Protectors		10-99: 40 cents each 100 or more: 25 cents each
Ashby Shift-Key Mod		$15.00
Lower-Case Display Encoder ROM		$25.00
Only Revision level 7 or later Apples.		
Books, Books, Books		compare our discount prices!
"Enhancing Your Apple II, vol. 1", Lancaster	($15.95)	$15.00
"Incredible Secret Money Machine", Lancaster	($7.95)	$7.50
"Micro Cookbook, vol. 1", Lancaster	($15.95)	$15.00
"Beneath Apple DOS", Worth & Lechner	($19.95)	$18.00
"Bag of Tricks", Worth & Lechner, with diskette	($39.95)	$36.00
"Apple Graphics & Arcade Game Design", Stanton	($19.95)	$18.00
"Assembly Lines: The Book", Roger Wagner	($19.95)	$18.00
"What's Where in the Apple", Second Edition	($24.95)	$23.00
"What's Where Guide" (updates first edition)	($9.95)	$9.00
"6502 Assembly Language Programming", Leventhal	($16.99)	$16.00
"6502 Subroutines", Leventhal	($12.99)	$12.00
"MICRO on the Apple--1", includes diskette	($24.95)	$23.00
"MICRO on the Apple--2", includes diskette	($24.95)	$23.00
"MICRO on the Apple--3", includes diskette	($24.95)	$23.00

Add $1 per book for US postage. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We take Master Charge, VISA and American Express ***

number of characters on the right end, including the decimal point. I save the "digits.after" parts of both strings, and also the maxima of the two parts. The maxima describe the result string (almost).

Lines 1900-2000 finish the description of the result string, by lengthening the integral (left) side by two characters. These two characters allow for extension of the result by carry, and for representation of the sign of the result using ten's complement notation. At this point I also clear the necessary bytes of the result to zero, so the buffer can be used as an accumulator.

Now comes the EASY part. Lines 2040-2100 add each operand in turn to the buffer contents. EASY. Just call the subroutine ADD.TO.BUFFER, and it's done! Don't worry, I'll amplify later.

In ten's complement notation, if the first digit is 0-4 the number is positive; if the first digit is 5-9, the number is negative. For example, 1234 looks like 001234; -1234 becomes 998766. Ten's complement means in decimal the same thing two's complement means in binary. I can form the ten's complement by subtracting the number from a power of ten equal to the number of digits in the result. In that example, 1000000-1234=998766. Note that the ten's complement is equal to the nine's complement plus one. (Since 10=9+1.)

Lines 2140-2410 convert the buffer contents from the ten's complement numeric notation back to ASCII. Lines 2140-2180 set or clear the CARRY and TENS.FLAG sign bits according to the first digit in the buffer. A negative number, with a first digit of 5-9, causes both of these variables to get a value of the form 1xxxxxxx.

Lines 2190-2360 scan through the number from right to left, making the ten's complement if the number was negative, and converting each digit to ASCII. Lines 2370-2400 store a minus sign in the first digit position if the result is negative.

Line 2410 calls a subroutine to chop off leading zeros, and move the minus sign if there is one. You may justifiably ask, "Why did you call a subroutine rather than use in-line code?" Because when I wrote it in-line, the local labels stretched out too far from the major label STRADD and caused an assembly error. Also, sometimes I use subroutines for clarity, even when the subroutine is only called once.

The final step is to pack the resulting string up and ship it to the result string variable. Lines 2450-2590 do just that. AS.GETSPA makes room at the bottom of string pool space, and AS.MOVSTR copies the string data. C'est finis!

Lines 2640-3100 do the actual addition. On entry, X is either 0 or 4, selecting either the first or second operand. SETUP.OPERAND copies the string address into VARPNT, and retrieves the length of the string. Lines 2690-2760 set or clear the TENS.FLAG and CARRY variables according to the sign of the operand.

Lines 2780-2810 compute the position in the buffer at which the operand will be aligned properly. We saved the size of the integral (left) side of the buffer in MAX.DIGITS.BEFORE. That plus the lenght of the fractional side of the operand tells us where this operand aligns. Since we are using ten's complement for negative numbers, rather than nine's complement, we don't have to worry about extending the fractional parts to the same length. We can just start adding at the end of the current operand. (In ten's complement form fractional extensions are zeros; in nine's complement form, the extension digits would all be nines.)

Lines 2830-3100 do the addition. X points into the buffer, and Y points into the operand string. To start with, both X and Y point just past the end; therefore the loop BEGINS with a test-and-decrement sequence. I first t-a-d the buffer pointer; if it is zero, all is finished. If not, on to t-a-d the string pointer. If it is zero, there are still digits left in the buffer, so I use an assumed leading zero digit for the operand. We still may have carries to propagate across the rest of the sum.

Assuming neither pointer is zero, line 2900 gets the next digit from the operand string. If it is a decimal point, I just store the decimal point ASCII value into the buffer. If you want to be able to ignore leading blanks, insert the following two lines between line 2920 and 2930:

```
2924        CMP #'           BLANK?
2925        BEQ .3           YES, USE ZERO.
```

I left them out in my version, because I forgot I promised it to you.

If the character is not a decimal point (or blank), it may be a minus sign or digit. I did not put any error checking in my program for other extraneous characters; if you try them, you will get extraneous results! I treat a sign as a leading zero in the arithmetic loop.

If the character is a digit, or an assumed leading zero, we can add it to the buffer's value. Lines 2960-3010 will complement the digit if the operand had a minus sign. Lines 3020-3070 add the current operand digit (or its complement) to the current buffer digit, plus any carry hung over from the preceding digit, and save the resulting carry in CARRY.

That's it! Now here is a short little Applesoft program to test the code.

```
100  REM TEST&+$,A$,B$
110  HIMEM: 36864: PRINT  CHR$ (4)"BLOAD B.STRING ADDER":
     CALL 36864
120  INPUT A$: INPUT B$
130  &  + $,A$,B$,C$
140  PRINT C$: GOTO 120
```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE *$50*

Is a trademark of Anthro-Digital, Inc.

Copyright © 1981
Written by John Rogers

See these programs at participating Computerland and other fine computer stores.

**Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202**

```
               1000 *SAVE S.SUPER STRING ADDER
               1010 *-------------------------------
               1020 *      STRING ADDITION:  & +$,A$,B$,C$
               1030 *-------------------------------
0200-          1040 BUFFER              .EQ $200 - $2FF
03F5-          1050 AMPERSAND.VECTOR    .EQ $3F5 - $3F7
00B1-          1060 AS.CHRGET           .EQ $00B1
DEC9-          1070 AS.SYNERR           .EQ $DEC9
DFE3-          1080 AS.PTRGET           .EQ $DFE3
DEBE-          1090 AS.CHKCOM           .EQ $DEBE
E452-          1100 AS.GETSPA           .EQ $E452
E5E2-          1110 AS.MOVSTR           .EQ $E5E2
               1120 *-------------------------------
               1130        .OR $9000
               1140        .TF B.STRING ADDER
               1150 *-------------------------------
9000- A9 4C    1160 SETUP  LDA #$4C     JMP OPCODE
9002- 8D F5 03 1170        STA AMPERSAND.VECTOR
9005- A9 21    1180        LDA #STRADD
9007- 8D F6 03 1190        STA AMPERSAND.VECTOR+1
900A- A9 90    1200        LDA /STRADD
900C- 8D F7 03 1210        STA AMPERSAND.VECTOR+2
900F- 60       1220        RTS
               1230 *-------------------------------
0071-          1240 FRESPC              .EQ $71,72
0083-          1250 VARPNT              .EQ $83,84
               1260 *-------------------------------
               1270 *      TWO SIMILAR BLOCKS, FOR A$ AND B$
               1280 *      REFERENCED WITH X=0 OR X=4
               1290 *-------------------------------
9010-          1300 A.LENGTH            .BS 1
9011-          1310 A.ADDR              .BS 2
9013-          1320 A.DIGITS.AFTER      .BS 1
               1330 *
9014-          1340 B.LENGTH            .BS 1
9015-          1350 B.ADDR              .BS 2
9017-          1360 B.DIGITS.AFTER      .BS 1
               1370 *-------------------------------
               1380 *      A THIRD BLOCK, NEARLY THE SAME AS ABOVE,
               1390 *      FOR C$:  REFERENCED WITH X=8
               1400 *-------------------------------
9018-          1410 C.LENGTH            .BS 1
9019-          1420 C.STRING            .BS 2
               1430 *-------------------------------
901B-          1440 CARRY               .BS 1
901C-          1450 TENS.FLAG           .BS 1
901D-          1460 C.ADDR              .BS 2
901F-          1470 MAX.DIGITS.BEFORE   .BS 1
9020-          1480 MAX.DIGITS.AFTER    .BS 1
               1490 *-------------------------------
               1500 *      & BRANCHES HERE
               1510 *-------------------------------
9021- C9 C8    1520 STRADD CMP #$C8     CHECK FOR "+$,"
9023- D0 0E    1530        BNE .1
9025- 20 B1 00 1540        JSR AS.CHRGET
9028- C9 24    1550        CMP #'$
902A- D0 07    1560        BNE .1
902C- 20 B1 00 1570        JSR AS.CHRGET
902F- C9 2C    1580        CMP #',
9031- F0 03    1590        BEQ .2
9033- 4C C9 DE 1600 .1     JMP AS.SYNERR
               1610 *-------------------------------
9036- A2 00    1620 .2     LDX #0       POINT AT A$ DATA
9038- 20 95 91 1630        JSR PARSE.STRING.NAME   FIRST OPERAND
903B- A2 04    1640        LDX #4       POINT AT B$ DATA
903D- 20 95 91 1650        JSR PARSE.STRING.NAME   SECOND OPERAND
9040- 20 BE DE 1660        JSR AS.CHKCOM           RESULT STRING
9043- 20 E3 DF 1670        JSR AS.PTRGET
9046- 8C 1A 90 1680        STY C.STRING+1          ADDRESS OF VARIABLE
9049- 8D 19 90 1690        STA C.STRING
```

```
                  1700 *--------------------------------
                  1710 *      SCAN BOTH STRINGS TO DETERMINE BUFFER PARAMETE
                  1720 *--------------------------------
904C- A2 00       1730        LDX #0       POINT AT A$ DATA
904E- 20 4E 91    1740        JSR SCAN     GET Y=LEFT LENGTH, X=RIGHT LENGTH
9051- 8E 13 90    1750        STX A.DIGITS.AFTER
9054- 8E 20 90    1760        STX MAX.DIGITS.AFTER
9057- 8C 1F 90    1770        STY MAX.DIGITS.BEFORE
905A- A2 04       1780        LDX #4       POINT AT B$ DATA
905C- 20 4E 91    1790        JSR SCAN     GET Y=LEFT LENGTH, X=RIGHT LEN
905F- 8E 17 90    1800        STX B.DIGITS.AFTER
9062- EC 20 90    1810        CPX MAX.DIGITS.AFTER
9065- 90 03       1820        BCC .3
9067- 8E 20 90    1830        STX MAX.DIGITS.AFTER
906A- CC 1F 90    1840 .3     CPY MAX.DIGITS.BEFORE
906D- 90 03       1850        BCC .4
906F- 8C 1F 90    1860        STY MAX.DIGITS.BEFORE
                  1870 *--------------------------------
                  1880 *      CLEAR THAT MUCH OF THE BUFFER
                  1890 *--------------------------------
9072- EE 1F 90    1900 .4     INC MAX.DIGITS.BEFORE  TWO MORE CHARS FOR
9075- EE 1F 90    1910        INC MAX.DIGITS.BEFORE    SIGN AND CARRY
9078- 18          1920        CLC
9079- AD 1F 90    1930        LDA MAX.DIGITS.BEFORE  TOTAL LENGTH OF RESULT
907C- 6D 20 90    1940        ADC MAX.DIGITS.AFTER
907F- 8D 18 90    1950        STA C.LENGTH
9082- A8          1960        TAY
9083- A9 00       1970        LDA #0       ZERO THE BUFFER FOR USE AS AN
9085- 99 FF 01    1980 .5     STA BUFFER-1,Y    ACCUMULATOR
9088- 88          1990        DEY
9089- D0 FA       2000        BNE .5
                  2010 *--------------------------------
                  2020 *      ADD A$ TO BUFFER
                  2030 *--------------------------------
908B- A2 00       2040        LDX #0       POINT AT A$ DATA
908D- 20 FA 90    2050        JSR ADD.TO.BUFFER
                  2060 *--------------------------------
                  2070 *      ADD B$ TO BUFFER
                  2080 *--------------------------------
9090- A2 04       2090        LDX #4       POINT AT B$ DATA
9092- 20 FA 90    2100        JSR ADD.TO.BUFFER
                  2110 *--------------------------------
                  2120 *      CONVERT BUFFER TO ASCII AGAIN
                  2130 *--------------------------------
9095- AD 00 02    2140        LDA BUFFER   SEE IF NUMBER IS NEGATIVE
9098- C9 05       2150        CMP #5       SET CARRY IF NEGATIVE, ELSE CLEAR
909A- 6A          2160        ROR          MAKE A=0XXXXXXX OR 1XXXXXXX
909B- 8D 1B 90    2170        STA CARRY        TO SET OR CLEAR THESE FLAGS
909E- 8D 1C 90    2180        STA TENS.FLAG    APPROPRIATELY
90A1- AE 18 90    2190        LDX C.LENGTH
90A4- F0 25       2200        BEQ .10      FINISHED
90A6- BD FF 01    2210 .6     LDA BUFFER-1,X
90A9- C9 2E       2220        CMP #'.
90AB- F0 18       2230        BEQ .9
90AD- 2C 1C 90    2240        BIT TENS.FLAG
90B0- 10 11       2250        BPL .8
90B2- 0E 1B 90    2260        ASL CARRY
90B5- A9 0A       2270        LDA #10
90B7- FD FF 01    2280        SBC BUFFER-1,X
90BA- C9 0A       2290        CMP #10
90BC- 90 02       2300        BCC .7
90BE- E9 0A       2310        SBC #10
90C0- 6E 1B 90    2320 .7     ROR CARRY
90C3- 09 30       2330 .8     ORA #'0
90C5- 9D FF 01    2340 .9     STA BUFFER-1,X
90C8- CA          2350        DEX
90C9- D0 DB       2360        BNE .6
90CB- 2C 1C 90    2370 .10    BIT TENS.FLAG   SEE ABOUT FINAL SIGN
90CE- 10 05       2380        BPL .11         VALUE IS POSITIVE
90D0- A9 2D       2390        LDA #'-         NEGATIVE, SO STUFF "-"
90D2- 8D 00 02    2400        STA BUFFER         IN FRONT OF BUFFER
90D5- 20 61 91    2410 .11    JSR CHOP.OFF.LEADING.ZEROES
```

```
               *-------------------------------
               *      PUT (BUFFER) IN OUTPUT STRING
               *-------------------------------
90D8- A2 08           LDX #8        POINT AT C$ DATA
90DA- 20 B3 91        JSR SETUP.OPERAND
90DD- 20 52 E4        JSR AS.GETSPA
90E0- A0 00           LDY #0
90E2- 91 83           STA (VARPNT),Y
90E4- C8              INY
90E5- A5 71           LDA FRESPC
90E7- 91 83           STA (VARPNT),Y
90E9- C8              INY
90EA- A5 72           LDA FRESPC+1
90EC- 91 83           STA (VARPNT),Y
90EE- AC 1E 90        LDY C.ADDR+1
90F1- AE 1D 90        LDX C.ADDR
90F4- AD 18 90        LDA C.LENGTH
90F7- 4C E2 E5        JMP AS.MOVSTR
               *-------------------------------
               *      ADD STRING TO BUFFER
               *      ENTER WITH X=0 FOR A$, X=4 FOR B$
               *-------------------------------
               ADD.TO.BUFFER
90FA- 20 B3 91        JSR SETUP.OPERAND
90FD- A8              TAY           STRING LENGTH
90FE- BD 13 90        LDA A.DIGITS.AFTER,X
9101- 48              PHA
9102- A2 00           LDX #0
9104- A1 83           LDA (VARPNT,X)  CHECK FOR MINUS SIGN
9106- C9 2D           CMP #'-
9108- F0 01           BEQ .1        YES, CARRY SET
910A- 18              CLC           ELSE CLEAR CARRY
910B- 6A       .1     ROR           MAKE A=0XXXXXXX OR 1XXXXXXX
910C- 8D 1C 90        STA TENS.FLAG  MAKE FLAGS<0 IF MINUS
910F- 8D 1B 90        STA CARRY
               *-------------------------------
9112- 18              CLC           POINT INTO BUFFER WHERE OPERAND
9113- 68              PLA                ALIGNS
9114- 6D 1F 90        ADC MAX.DIGITS.BEFORE
9117- AA              TAX
               *-------------------------------
9118- 8A       .2     TXA           TEST X FOR BEGINNING OF BUFFER
9119- F0 32           BEQ .8        YES, FINISHED!
911B- CA              DEX           NO, BACK ANOTHER ONE
911C- 98              TYA           CHECK OPERAND POINTER
911D- F0 0B           BEQ .3        END OF OPERAND, BUT WE
               *                          STILL NEED TO FINISH
911F- 88              DEY           BACK UP IN OPERAND      CARRIES
9120- B1 83           LDA (VARPNT),Y  NEXT CHAR FROM OPERAND
9122- C9 2E           CMP #'.       DECIMAL POINT?
9124- F0 21           BEQ .7        YES, SKIP OVER IT
9126- C9 2D           CMP #'-       MINUS SIGN?
9128- D0 02           BNE .4        NO, MUST BE DIGIT
912A- A9 30    .3     LDA #'0       ASCII ZERO THEN
912C- 29 0F    .4     AND #$0F      CONVERT ASCII TO BINARY
912E- 2C 1C 90        BIT TENS.FLAG
9131- 10 05           BPL .5        NOT 9'S COMPLEMENTING
9133- 49 FF           EOR #$FF
9135- 18              CLC
9136- 69 0A           ADC #10       FORM 9'S COMPLEMENT
9138- 0E 1B 90 .5     ASL CARRY     GET PREVIOUS CARRY INTO C-BIT
913B- 7D 00 02        ADC BUFFER,X
913E- C9 0A           CMP #10       SEE IF CARRY
9140- 90 02           BCC .6        NO
9142- E9 0A           SBC #10       YES, BACK THIS DIGIT DOWN
9144- 6E 1B 90 .6     ROR CARRY     SAVE CARRY FOR NEXT LOOP
9147- 9D 00 02 .7     STA BUFFER,X
914A- 4C 18 91        JMP .2
914D- 60       .8     RTS
               *-------------------------------
               *      SCAN STRING
               *      ENTER WITH X=0 FOR A$, X=4 FOR B$
               *      RETURN WITH X = # DIGITS AFTER DECIMAL POINT
               *                  (COUNTING THE DECIMAL POINT
               *            Y = # DIGITS BEFORE DECIMAL POINT
               *                  (COUNTING SIGN IF ANY)
               *-------------------------------
```

S-C Macro Cross Assemblers

The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various popular microprocessors. Combining the versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system. Hobbyists and engineers alike will find the friendly combination the easiest and best way to extend their skills to other microprocessors.

The S-C Macro Cross Assemblers are all identical in operation to the S-C Macro Assembler; only the language assembled is different. They are sold as upgrade packages to the S-C Macro Assembler. The S-C Macro Assembler, complete with 100-page reference manual, costs $80; once you have it, you may add as many Cross Assemblers as you wish at a nominal price. The following S-C Macro Cross Assembler versions are now available, or soon will be:

Motorola:	6800/6801/6802	now	$32.50
	6805	now	$32.50
	6809	now	$32.50
	68000	now	$50
Intel:	8048	now	$32.50
	8051	soon	$32.50
	8085	soon	$32.50
Zilog:	Z-80	now	$32.50
RCA:	1802/1805	soon	$32.50
Rockwell:	65C02	now	$20

The S-C Macro Assembler family is well known for its ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependablility, and user-friendliness. There are 20 assembler directives to provide powerful macros, conditional assembly, and flexible data generation. INCLUDE and TARGET FILE capabilities allow source programs to be as large as your disk space. The integrated, co-resident source program editor provides global search and replace, move, and edit. The EDIT command has 15 sub-commands combined with global selection.

Each S-C Assembler diskette contains two complete ready-to-run assemblers: one is for execution in the mother-board RAM; the other executes in a 16K RAM Card. The HELLO program offers menu selection to load the version you desire. The disks may be copied using any standard Apple disk copy program, and copies of the assembler may be BSAVEd on your working disks.

S-C Software Corporation has frequently been commended for outstanding support: competent telephone help, a monthly (by subscription) newsletter, continuing enhancements, and excellent upgrade policies.

S-C Software Corporation (214) 324-2050
P.O. Box 280300, Dallas, Texas, 75228

```
                SCAN
914E- 20 B3 91          JSR SETUP.OPERAND
9151- A0 00             LDY #0
9153- AA                TAX
9154- F0 0A             BEQ .2       NULL STRING
9156- B1 83     .1      LDA (VARPNT),Y
9158- C9 2E             CMP #'.      LOOKING FOR DECIMAL POINT
915A- F0 04             BEQ .2
915C- C8                INY
915D- CA                DEX
915E- D0 F6             BNE .1
9160- 60        .2      RTS
                *--------------------------------
                *       CHOP OFF LEADING ZEROES
                *--------------------------------
                CHOP.OFF.LEADING.ZEROES
9161- A0 01             LDY #1       FIND FIRST NON-ZERO POSITION
9163- B9 00 02  .1      LDA BUFFER,Y
9166- C9 30             CMP #'0
9168- D0 07             BNE .2
916A- C8                INY
916B- CC 1F 90          CPY MAX.DIGITS.BEFORE
916E- 90 F3             BCC .1
9170- 88                DEY
9171- AD 00 02  .2      LDA BUFFER   SIGN, MAYBE
9174- C9 2D             CMP #'-
9176- D0 04             BNE .3
9178- 88                DEY
9179- 99 00 02          STA BUFFER,Y
917C- 18        .3      CLC
917D- 98                TYA
917E- 69 00             ADC #BUFFER
9180- 8D 1D 90          STA C.ADDR
9183- A9 00             LDA #0
9185- 69 02             ADC /BUFFER
9187- 8D 1E 90          STA C.ADDR+1
918A- 38                SEC
918B- 98                TYA
918C- 49 FF             EOR #$FF
918E- 6D 18 90          ADC C.LENGTH
9191- 8D 18 90          STA C.LENGTH
9194- 60                RTS
                *--------------------------------
                *       PARSE STRING NAME, SET UP POINTER
                *--------------------------------
                PARSE.STRING.NAME
9195- 8A                TXA
9196- 48                PHA
9197- 20 BE DE          JSR AS.CHKCOM
919A- 20 E3 DF          JSR AS.PTRGET     GET SECOND STRING PNTR
919D- 68                PLA
919E- AA                TAX
919F- A0 00             LDY #0
91A1- B1 83             LDA (VARPNT),Y    GET LENGTH
91A3- 9D 10 90          STA A.LENGTH,X
91A6- C8                INY
91A7- B1 83             LDA (VARPNT),Y    GET ADDRESS OF DATA
91A9- 9D 11 90          STA A.ADDR,X
91AC- C8                INY
91AD- B1 83             LDA (VARPNT),Y
91AF- 9D 12 90          STA A.ADDR+1,X
91B2- 60                RTS
                *--------------------------------
                *       LOAD ADDRESS INTO VARPNT
                *       X=0 FOR A$, X=4 FOR B$
                *--------------------------------
                SETUP.OPERAND
91B3- BD 11 90          LDA A.ADDR,X
91B6- 85 83             STA VARPNT
91B8- BD 12 90          LDA A.ADDR+1,X
91BB- 85 84             STA VARPNT+1
91BD- BD 10 90          LDA A.LENGTH,X
91C0- 60                RTS
                *--------------------------------
```

More on the Macro-Videx Connection................Bill Linn

Don Taylor's original article in the August (1982) issue of AAL and Mike Laumer's follow-up the next month gave us the patches for running the S-C Macro Assembler in conjunction with the Videx 80-column board. I recently purchased a Videx card in order to implement the 80-column version of ES-CAPE, so I installed the patches.

I have really enjoyed using the Macro assembler in 80-column mode. Naturally, though, I couldn't resist adding a few enhancements to Don's and Mike's work.

Mike added the right arrow code, which copies characters off the Videx screen, but he stopped short of implementing the Escape-L LOAD sequence. To install the following code, you will need to change line 3080 in Don's article to point to my routine. Change it to "3080   .DA MY.ESC.L-1". Also, the STX instruction at line 4235 in Mike's article must be labelled GETCH.

```
*---------------------------------------
SCM.INSTALL .EQ SCM.BASE+$52A
*
MY.ESC.L
       CPX #0          CURSOR AT BEGINNING?
       BEQ .1          YES, CONTINUE
       JMP SCM.ESC.L   NO, LET S-C HANDLE IT
.1     LDA #0          CONNECT DOS
       STA $AA52       BY SETTING INTERCEPT STATE = 0
       LDA #$84        SEND A CTRL-D
       JSR MON.COUT
.2     LDA LOADCMD,X
       JSR SCM.INSTALL
       JSR FAKE.COUT
       CPX #6
       BCC .2
.3     STX $406        SAVE CHAR POS'N
       JSR GETCH       GET SCREEN CHAR
       LDX $406        RESTORE POS'N
       JSR SCM.INSTALL
       JSR FAKE.COUT
       CPX #40         40 CHARS SENT YET?
       BNE .3          NO, LOOP BACK
       JMP CLREOP      CLEAR TO END OF PAGE
*                      AND EXIT
*
LOADCMD    .AS -/LOAD  /
*---------------------------------------
```

Secondly, I wanted a longer "*---" line on my screen, so I changed it to 68 characters instead of 38. This uses more of the 80 column screen, without wrapping around during assembly. To make this modification insert the following two lines after the label "INSTALL.PATCHES" in Don's original listing:

```
       LDA #68
       STA SCM.BASE+$494
```

Finally, I changed the dimensions of the Videx cursor so that it looks like a blinking underline instead of a blinking block. (Users of my ES-CAPE are already familiar with my love for the blinking underline!) Insert the following lines immediately after the "INSTALL.VECTORS" label:

```
     LDA #$0A       VIDEX REGISTER 10
     STA V.DEV0
     LDA #$68
     STA V.DEV0+1
     LDA #$0B       VIDEX REGISTER 11
     STA V.DEV0
     LDA #$08
     STA V.DEV0+1
```

Speaking of ES-CAPE, I am making progress on Version 2 and have included suggestions from many of you. If you have others, please drop me a line soon at 3199 Hammock Creek, Lithonia, GA 30058, or call evenings at (404) 483-7637.

DISASM (Version 2.2) $30.00

Use DISASM, the intelligent disassembler, to convert 6502 machine code into meaningful, symbolic source. It creates a text file which is directly compatable with DOS ToolKit, LISA and S-C (both 4.0 & Macro) Assemblers. Use DISASM to customize existing machine language programs to your own needs or just to see how they work. DISASM handles multiple data tables, invalid op codes and displaced object code (the program being disassembled doesn't have to reside in the memory space in which it executes). DISASM lets you even substitute MEANINGFUL labels of your own choice (100 commonly used Monitor & Pg Zero names included in Source form to get you rolling). The address-based cross reference table option results in either a selective or complete cross reference (to either screen or printer). Page Zero and External references are listed separately in numeric order. The cross reference table provides as much insight into the inner workings of machine language programs as the disassembly itself. DISASM has proven to be an invaluable aid for both the novice and expert alike.

Utilities For Your S-C Assembler (4.0)

SC.GSR: A Global Search and Replace Eliminates Tedious Manual Renaming Of Labels................$20.00
SC.XREF: A Linenumber-Based Global Cross Reference Table For Complete Source Documentation......$20.00
SC.TAB: Tabulates Source Files Into Neat, Readable Form. Encourages Fast, Free-Format Entry....$15.00
SC UTILITY PAK: Includes All Three Utilities Described Above (You Save $10.00).................$45.00

All of the above programs are written entriely in machine language and are provided on a standard 3.3 DOS formatted diskette.

Avoid A $3.00 Shipping/Handling Charge By Mailing Full Payment With Order

RAK-WARE
41 Ralph Road
West Orange NJ 07052

***** SAY YOU SAW IT IN 'APPLE ASSEMBLY LINE'! *****

On CATALOG ARRANGER and RAM Card DOS

Chuck Welman just called to report some errors in the January piece on using CATALOG ARRANGER with a relocated DOS. He says that the sentence about where to put the BIT MONREAD statements had problems. Here's his corrected version:

"Then add BIT MONREAD at these positions: Lines 1675, 3775, 3895, 3955, 4015 (".5" moved to this line), 4205 (".3" moved to this line, 4315, 4425, 4455 (".7" moved to this line), and 4895."

Chuck also passed along instructions for using FILENAME EDITOR with a RAM Card DOS. Here are his additions:

```
     2635 .3      BIT MONREAD
     2640         JSR MON.BELL
     2642         BIT DOSREAD
     2644         BIT DOSREAD
     2646         RTS
```

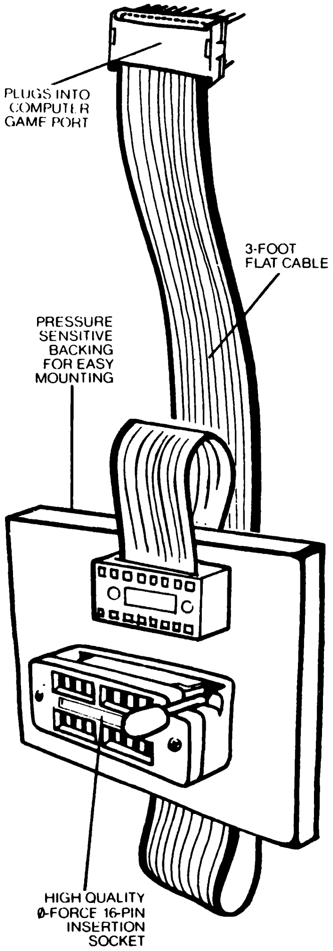
Thanks to all of you for showing your appreciation for these programs.

Quickie No. 6.....Bob Sander-Cederlof

Here is a little run-anywhere program sure to wake up the neighborhood dogs. Put it in your program as a last resort to get attention, because the only escape is by RESET or power-off.

```
1000 ALARM  INY         INCREMENT DELAY
TIME
1010        TYA
1020        TAX         DELAY COUNT TO X
1030        LDA $C030 TOGGLE SPEAKER
1040 .1     DEX         DELAY LOOP
1050        BNE .1
1060        BEQ ALARM ....FOREVER....
```

That's it, only eleven bytes! For a slightly different effect, change the "DEX" in line 1030 to "INX".



- **Extends game port outside computer with 3-foot cable.**
- **Prolongs life of game accessories with Ø-Force insertion socket.**
- **Mounts anywhere in seconds with pressure sensitive backing.**
- **Offers better performance and reliability with high grade components.**

Patch to Fix .TI Problem........................Mike Laumer

You may have noticed the annoying problem with the .TI directive, in which there is sometimes a blank line after the title line and sometimes not. The blank line is there when the page break is forced with a .PG directive, but not when it is caused by merely filling a page.

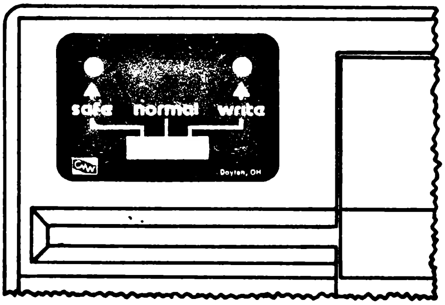
The following little patch will fix it. I haven't put a definite address on the patch, because I don't know what other patches you may already have appended to the assembler. Just find an empty place and plop it in!

```
Motherboard version:    :$21F0:4C xx yy   (was 20 CF 2C)
                        :$yyxx:20 CF 2C 4C E3 21

RAM Card version:       :$E33C:4C xx yy   (was 20 1B EE)
                        :$yyxx:20 1B EE 4C 2F E3
```

Another .TI problem of which I am aware is that the line count is messed up on the first page of the symbol table listing. This is caused by the fact that the extra carriage returns in the "SYMBOL TABLE" message are not counted. You can clean up the appearance by making the last line of your source program be ".PG"; this forces the symbol table to start on a fresh page.

'Oops-Proof' Your Disk][



"OOPS!! "#!#?"!! I just wrote over a valuable disk file (forgot to put on a write-protect tab)". If this hasn't happened to you, it's just a matter of time! A CMW Disk Switch® will help prevent these accidents, and will give you new features, too. It provides manual control and positive indication of your disk drive's write protect status. Switch to "SAFE" mode to prevent writing, even without a tab. Switch to "WRITE" mode to override a tab or write to an unnotched diskette. Select "NORMAL" mode for normal tab sensing. Model DSA-2 consists of a Disk Switch installed in a replacement Disk][® faceplate. It installs in minutes, and requires only five easy solder connections.

OTHER FINE CMW PRODUCTS

- The PROM Switch - Model PS-1
  Switch 1 of 3 ROMs/EPROMS into D0-F8.
- "PROMETTE"® EPROM Carrier - Model PC-1
  Adapts a 2716 EPROM to D0-F8 sockets.

Write or call for a catalog.

THE DISK SWITCH

- Total control of write protect status
- Positive status indication (Green/Red LEDs)
- Attractive switchplate matches drive decor
- Fully assembled and tested
- Installs in minutes; full instructions
- Drive easily restored to original condition

Model DSA-2 (w/Apple logo front)	$39.95
Model DSA-1 (w/plain front)	$34.95
Model DSK-1 (kit of parts; installs in your drive faceplate)	$11.95

Models for non-Apple drives: write CMW.

Add $2.00 - shipping   OH add 5.5% sales tax
Visa, MC, MO, Check, COD (add $1.40)
Disk][is a trademark of Apple Computer Inc

COMPUTER MICRO WORKS, INC
P.O. Box 33651   Dayton, Ohio 45433
(305) 777-0268 (FL sales office)

Apple //e Notes............................Bob Sander-Cederlof

We don't have one yet, but we did play with one for about an hour last week. All our software works fine, as long as you stay in the 40-column caps-lock mode. We will be making new versions available in the near future which take full advantage of the extended memory, lower-case, and 80-column display.

The best write-up I have seen yet on the //e is in the February 1983 Apple Orchard (published by the International Apple Core, 908 George St., Santa Clara, CA 95050).

Here are some of the things that caught my attention:

* Real shift key, and a caps-lock key.

* Open-Apple and Closed-Apple keys, which duplicate the first two paddle buttons.

* Recessed RESET key. CTRL-RESET required (no longer a switchable option). CTRL-Closed-Apple-RESET starts a memory test program.

* Two 8K ROMs, instead of six 2K ROMs. The extra 2K of ROM space is used by the modified Monitor program. Fancy soft-switches map the extra 2K into the $C000-C7FF space. These sockets are supposedly compatible with 2764 EPROMs.

* Apparently the Monitor now uses (clobbers) zero-page locations $08 and $1F.

* Up- and down-arrows on the keyboard. Down is CTRL-J, or linefeed. Up is CTRL-K.

* The keyboard includes all the ASCII set, even $7F (DELETE, or RUBOUT).

* 64K RAM on the motherboard. This simulates an Apple II Plus with a 16K RAM card in slot 0.

* New slot instead of slot 0, with 60-pin connector (other slots still have 50-pin connectors). Apple's 80-column card plugs in here. The extra pins carry other signals not normally available at the slots. Look for some amazing new combined function cards from the peripheral-card makers for this slot! I wouldn't be surprised to find ads real soon for 256K RAM cards including 80-column support, clock-calendar, serial/parallel interfaces, and all on one card.

* 80-column card with or without extra 64K RAM. But this 64K RAM is soft-switched in a totally different manner. It maps over the same space as the motherboard 64K, with switches to map portions such as page-zero, text screen, hi-res screen, and so on.

- Now you can READ the state of most of the soft-switches. Bit 7 (high bit) tells the state, as follows:

$C013	--	RAMREAD
$C014	--	RAMWRT
$C015	--	SLOTCXROM/CX00ROM
$C016	--	ALTZP/MAIN
$C017	--	SLOTC3ROM/SLOTROM
$C018	--	80 COL STORE
$C019	--	VERTICAL BLANKING
$C01A	--	TEXT
$C01B	--	MIXED MODE
$C01C	--	PAGE2
$C01D	--	HIRES
$C01E	--	ALTCHAR
$C01F	--	80 COL DISP

- Yes, you saw right...the vertical blanking signal is now readable! So lovers of Lancaster's Enhancements can continue to tinker!

- Inverse lower-case display is selectable, at the expense of the flashing mode.

- The cursor display is different. A small checkerboard alternates with the character under the cursor in 40-column mode. In 80-column mode an inverse blank is the normal cursor, and an inverse "+" is used when in escape-mode.

Whether we view the changes as improvements or not, the //e will very soon be the standard we all have to deal with. The same situation arose when Apple switched from II to II Plus. A year from now, when 300,000 have been sold, we will wonder how we ever lived without it!

Macro Assembler Patch

Peter Bartlett, of Chicago, has reported an unpublished limit on the number of Target Files that can be generated by one assembly. Right now there can only be 31; above that number the load address and length bytes go astray. If you need more than 31 files from one assembly, you can make the following patches:

Regular version

```
          :$29EA:3F
```

Language Card version

```
          :$C083 C083 EB36:3F N C080
```

These patches will allow you to have up to 63 target files. That should be plenty!

TRAPPER: An Applesoft Input Tuner..............Allen Marsalis

How would you like a radio which played every available station at one time? Well that's how I sometimes feel about using Applesoft's INPUT statement. I want to be able to "tune in" on the character(s) of the input stream, in much the same way as a radio tunes into a station. Applesoft's INPUT statement, however, accepts all characters typed into the keyboard and allows up to 255 of them. This means that I have to do a lot of checking and monitoring of string lengths and characters to avoid input errors.

For example, when answering a Y or N question, what happens when the user inputs "WXYZ"? Provisions are needed within the program to guard against such errors. This can be very inconvenient and space-consuming, yet it is essential for good programming.

A better example occurs when you are creating a disk file. Field lengths and data types are often restricted, such as in a name, address, or social security number. A SSN, for instance, has a fixed length and must be constructed of numbers only. Checking a field such as this can be very time consuming and lengthy. In fact, it seems that a quarter of the contents of my Applesoft programs does nothing but check on field lengths, option boundaries, and other input checks.

So, I set out to create an input routine which would allow Applesoft to "tune" into the characters specified and also monitor the field length. I've seen several input routines such as this on larger systems, but all had one disadvantage: Only a fixed number of options were available, such as alpha only, numeric only, and (Y or N) input. More options available meant more parameters were necessary, making the systems more cumbersome to work with. After much thought I decided on a totally new approach which would allow almost limitless control of input. I christened this routine TRAPPER for "Tuning and Regulating APPlesoft Entries by Restriction."

TRAPPER employs a coded restriction string (not unlike Applesoft's IF expression) to tune out the characters I don't want to accept. TRAPPER is then, in essence, a tiny interactive interpreter that provides a short, convenient method of filtering out any unwanted characters in the input. Here's how it works.

TRAPPER uses three parameters as follows:

```
Syntax:  & INPUT (A, B$, C$)
     A:  Input field length (real expression)
    B$:  Coded restriction string (string expression)
         includes:  > < = ' AND OR NOT <sp> <single char>
    C$:  Input string (string variable)
         variable to receive input
```

As I have said, the restriction string is a simple relational expression as is used by Applesoft's IF statement. It is constructed of the following special characters and rules:

1) < > = are its relational operators
2) AND OR NOT are its logical operators
3) Blanks are allowed anywhere within the expression, but lengthy expressions increase the delay between keystrokes.
4) One and only one character is allowed within single quotes.
5) <cr> and <-- have special functions and cannot be trapped.
6) Parentheses are not yet implemented.

EXAMPLES:

```
    YN$ = " ='Y' OR ='N' "               :REM (Y OR N) ONLY
    NOSP$ = " NOT =' ' "                 :REM NO SPACES ALLOWED
    MENU$ = " NOT <'1' AND NOT >'4' "    :REM ALLOWS 1 THRU 4
    WAITCR$ = ""                         :REM WAIT FOR A <CR>
```

After using Trapper awhile, I noticed a significant reduction in the size of my Applesoft programs, with even better error trapping than ever before possible. And it doesn't print that leading question mark which I never did like (not all input prompts are questions.)

For a 48K Apple, DOS sets HIMEM at $9600. Trapper resides just below this at $9300 and moves HIMEM down to that point.

```
                  1000 *SAVE S.TRAPPER
                  1010 *--------------------------------
                  1020 *       TRAPPER, BY ALLEN MARSALIS
                  1030 *--------------------------------
                  1040        .OR $9300
                  1050        .TF B.TRAPPER
                  1060 *--------------------------------
001A-             1070 RLEN   .EQ $1A        RESTRICTION STRING
001B-             1080 RSTR   .EQ $1B        DESCRIPTOR
0052-             1090 TEMPPT .EQ $52
0053-             1100 LASTPT .EQ $53
0071-             1110 FRESPC .EQ $71,72
0073-             1120 HIMEM  .EQ $73,74
0083-             1130 VARPNT .EQ $83,84
00A0-             1140 FACMO  .EQ $A0
                  1150 *--------------------------------
0200-             1160 BUF    .EQ $200       INPUT BUFFER
03F5-             1170 AMPVEC .EQ $3F5       AMPERSAND VECTOR
C010-             1180 STROBE .EQ $C010      KEYBOARD STROBE
                  1190 *--------------------------------
DD67-             1200 AS.FRMNUM   .EQ $DD67     EVALUATE NUMERIC FORMULA
DD6C-             1210 AS.CHKSTR   .EQ $DD6C     REQUIRE STRING
DD7B-             1220 AS.FRMEVL   .EQ $DD7B     EVALUATE GENERAL FORMULA
DEB8-             1230 AS.CHKCLS   .EQ $DEB8     REQUIRE ")"
DEBE-             1240 AS.CHKCOM   .EQ $DEBE     REQUIRE ","
DEBB-             1250 AS.CHKOPN   .EQ $DEBB     REQUIRE "("
DEC0-             1260 AS.SYNCHR   .EQ $DEC0     REQUIRE (A-REG)
DEC9-             1270 AS.SYNERR   .EQ $DEC9     SYNTAX ERROR
DFE3-             1280 AS.PTRGET   .EQ $DFE3     GET VARIABLE PNTR
E452-             1290 AS.GETSPA   .EQ $E452     GET SPACE IN STRING AREA
E5E2-             1300 AS.MOVSTR   .EQ $E5E2     COPY STRING DATA
E604-             1310 AS.FRETMP   .EQ $E604     FREE TEMPORARY STRING
E6FB-             1320 AS.CONINT   .EQ $E6FB     CONVERT FAC TO 8-BITS
```

Help Wanted?

Assistant Programmer™

Menu Generator and Screen Text Editor

Accepts all characters from keyboard (including " [/ _ ").
Shifts individual lines or entire page with I-J-K-M keys.
Splits a line or combines two lines at cursor position.
I nserts input or variable print commands at cursor position.
Spaces added or deleted between lines with a single keystroke.
Text file created to "EXEC" menu (BASIC) into your program.
Accents your titles with borders, using minimum amount of bytes.
No disk swapping, entire program resides in memory for easy access.
Text file created that reinstalls screens for future development.

Prints copy of menu in screen format with row and column edge scales.
Renders line at cursor position inverse or flashing with keystroke.
Overhaul your existing menu with transparent capture routine.
Gives printed list of input/print commands with row & column numbers.
Right, left and center justification of menu at cursor position.
All features and procedures fully documented and tutored.
Menus can be designed on the screen, no need for graph paper layouts.
Makes horizontal lines print in vertical format at the touch of a key.
Error messages displayed for all options (display voidable).
Return coupon for your ticket to fast, efficient and friendly menus.

ALL FOR ONLY $49.95.

Requirements: Apple II Plus* with 48K & disc drive.
*Apple II is a registered trademark of
Apple Computer, Inc.

UNPROTECTED

ALSO AVAILABLE NOW!

Barney Stone's

MICRO MEMO II **calendar/tickle file/phone listing**
Improving the Proven: completely compatible with Micro Memo
$49.95

RUSH □ Assistant Programmer,
□ Micro Memo II, at once!
□ Enclosed is check or money order for $49.95 each.*
□ Please ship COD (add $3.00 for COD charge).
Dealer inquiries welcome. *Pa. residents add 6% sales tax.

Soph-Key
P.O. Box 387
Flourtown PA 19031
(215) 836-1056

NAME ______________________

ADDRESS ______________________

CITY ______________ STATE ______ ZIP ______

```
                    1330 *--------------------------------
FC9C-               1340 MON.CLREOL .EQ $FC9C     CLEAR TO END-OF-LINE
FD0C-               1350 MON.RDKEY  .EQ $FD0C     READ A KEY
FDED-               1360 MON.COUT   .EQ $FDED     DISPLAY A CHARACTER
                    1370 *--------------------------------
9300- A9 4C         1380 SETUP  LDA #$4C     "JMP" OPCODE
9302- 8D F5 03      1390        STA AMPVEC
9305- A9 18         1400        LDA #TRAPPER
9307- 8D F6 03      1410        STA AMPVEC+1
930A- A9 93         1420        LDA /TRAPPER
930C- 8D F7 03      1430        STA AMPVEC+2
930F- A9 00         1440        LDA #SETUP   SET HIMEM UNDER TRAPPER
9311- 85 73         1450        STA HIMEM
9313- A9 93         1460        LDA /SETUP
9315- 85 74         1470        STA HIMEM+1
9317- 60            1480        RTS
                    1490 *--------------------------------
                    1500 *      AMPERSAND COMES HERE
                    1510 *--------------------------------
                    1520 TRAPPER
9318- A9 84         1530        LDA #$84          "INPUT" TOKEN
931A- 20 C0 DE      1540        JSR AS.SYNCHR
931D- 20 BB DE      1550        JSR AS.CHKOPN     "& INPUT ("
9320- 20 67 DD      1560        JSR AS.FRMNUM     READ FIELD LENGTH PARAMETER
9323- 20 FB E6      1570        JSR AS.CONINT     CONVERT TO 8-BIT VALUE
9326- 8E DE 94      1580        STX FL            SAVE FIELD LENGTH
9329- 20 BE DE      1590        JSR AS.CHKCOM     ","
932C- 20 7B DD      1600        JSR AS.FRMEVL     GET RESTRICTION STRING
932F- 20 6C DD      1610        JSR AS.CHKSTR
9332- 20 BE DE      1620        JSR AS.CHKCOM     ANOTHER ","
9335- A0 02         1630        LDY #2            SAVE DESCRIPTOR
9337- B1 A0         1640 .1     LDA (FACMO),Y
9339- 99 1A 00      1650        STA RLEN,Y
933C- 88            1660        DEY
933D- 10 F8         1670        BPL .1
933F- A5 52         1680        LDA TEMPPT        DID FRMEVL MAKE A TEMP
9341- C9 56         1690        CMP #$56                               STRING?
9343- 90 07         1700        BCC .2            NO
9345- A5 53         1710        LDA LASTPT        YES, SO FREE THE TEMP
9347- A0 00         1720        LDY #0
9349- 20 04 E6      1730        JSR AS.FRETMP
934C- A9 00         1740 .2     LDA #0            INIT BUFFER INDEX
934E- 8D E2 94      1750        STA BINDEX
                    1760 *---UNDERSCORE INPUT FIELD-------
9351- A9 DF         1770        LDA #$DF          UNDERLINE CHAR
9353- 20 D1 94      1780        JSR PRINT.FIELD
9356- A9 88         1790        LDA #$88          BACKSPACE  TO BEGINNING
9358- 20 D1 94      1800        JSR PRINT.FIELD
                    1810 *---READ A KEY-------------------
935B- 2C 10 C0      1820        BIT STROBE        DON'T ALLOW TYPE AHEAD
935E- 20 0C FD      1830 .3     JSR MON.RDKEY     READ NEXT KEY
9361- 29 7F         1840        AND #$7F          INTERNAL FORM
9363- 8D E1 94      1850        STA KEY           SAVE IT
                    1860 *---BACKSPACE--------------------
9366- C9 08         1870        CMP #$08          BACKSPACE?
9368- D0 1A         1880        BNE .22           NO
936A- AD E2 94      1890        LDA BINDEX        IGNORE AT BEGINNING OF LINE
936D- F0 12         1900        BEQ .21
936F- A9 88         1910        LDA #$88          YES, ECHO IT
9371- 20 ED FD      1920        JSR MON.COUT
9374- A9 DF         1930        LDA #$DF          REPLACE UNDERLINE
9376- 20 ED FD      1940        JSR MON.COUT
9379- A9 88         1950        LDA #$88          BACKSPACE AGAIN
937B- 20 ED FD      1960        JSR MON.COUT
937E- CE E2 94      1970        DEC BINDEX        BACK UP BUFFER TOO
9381- 4C 5E 93      1980 .21    JMP .3
                    1990 *---CARRIAGE RETURN--------------
9384- C9 0D         2000 .22    CMP #$0D          RETURN?
9386- D0 27         2010        BNE .23           NO
9388- 20 9C FC      2020        JSR MON.CLREOL
938B- 20 E3 DF      2030        JSR AS.PTRGET     GET DESTINATION STRING
938E- 20 B8 DE      2040        JSR AS.CHKCLS     MUST HAVE ")" AT END
9391- AD E2 94      2050        LDA BINDEX        LENGTH OF INPUT LINE
9394- 20 52 E4      2060        JSR AS.GETSPA     FIND ROOM FOR IT
9397- A0 00         2070        LDY #0            MOVE IN DESCRIPTOR
9399- 91 83         2080        STA (VARPNT),Y
939B- C8            2090        INY
939C- A5 71         2100        LDA FRESPC
939E- 91 83         2110        STA (VARPNT),Y
93A0- C8            2120        INY
93A1- A5 72         2130        LDA FRESPC+1
93A3- 91 83         2140        STA (VARPNT),Y
```

```
93A5- A0 02            LDY /BUF         COPY DATA INTO STRING
93A7- A2 00            LDX #BUF
93A9- AD E2 94         LDA BINDEX
93AC- 4C E2 E5         JMP AS.MOVSTR    ...AND RETURN
                *---CHECK IF VALID KEY-----------
93AF- 20 E0 93  .23    JSR CHECK.RESTRICTIONS
                *---CHECK VALIDITY AND ECHO------
93B2- AD E1 94         LDA KEY          GET KEY AGAIN
93B5- AD E2 94         LDA BINDEX
93B8- CD DE 94         CMP FL
93BB- B0 1F            BCS .27          TOO FAR, ABORT KEY
93BD- AD DF 94         LDA NEW          IF NEW = FAIL, ABORT KEY
93C0- F0 1A            BEQ .27          YES, ABORT KEY
93C2- AD E1 94         LDA KEY
93C5- AC E2 94         LDY BINDEX
93C8- 99 00 02         STA BUF,Y        PUT KEY INTO BUFFER
93CB- EE E2 94         INC BINDEX
93CE- C9 20            CMP #$20         IF KEY WAS CONTROL-KEY,
93D0- B0 02            BCS .26             THEN PRINT SPACE
93D2- A9 20            LDA #$20
93D4- 09 80     .26    ORA #$80
93D6- 20 ED FD         JSR MON.COUT     ECHO
93D9- 4C 5E 93         JMP .3           NEXT KEY
93DC- A9 07     .27    LDA #$07         RING BELL
93DE- D0 F4            BNE .26
                *--------------------------------
                CHECK.RESTRICTIONS
93E0- A9 00            LDA #0
93E2- 8D E3 94         STA RINDEX       RINDEX = 0
93E5- 8D DF 94         STA NEW          NEW = FAIL
93E8- 8D DD 94         STA ANDOR        ANDOR = OR
93EB- 8D DC 94         STA NOT          NOT = FALSE
                *---FETCH OPERATOR---------------
93EE- AC E3 94  .4     LDY RINDEX       IF RINDEX >= RLEN,
93F1- C4 1A            CPY RLEN             THEN QUIT SCAN
93F3- 90 01            BCC .5           NOT YET
93F5- 60               RTS
93F6- B1 1B     .5     LDA (RSTR),Y     FETCH OPERATOR
93F8- EE E3 94         INC RINDEX
                *---DETERMINE OPERATION----------
93FB- C9 20            CMP #'           IGNORE BLANKS
93FD- F0 EF            BEQ .4
93FF- C9 3C            CMP #'<          < = >, THEN FETCH OPERAND
9401- F0 45            BEQ .10
9403- C9 3E            CMP #'>
9405- F0 41            BEQ .10
9407- C9 3D            CMP #'=
9409- F0 3D            BEQ .10
940B- C9 41            CMP #'A          "AND"
940D- F0 0B            BEQ .7
940F- C9 4F            CMP #'O
9411- F0 18            BEQ .8
9413- C9 4E            CMP #'N          "NOT"
9415- F0 20            BEQ .9
9417- 4C C9 DE         JMP AS.SYNERR
                *---AND OPERATOR-----------------
941A- A9 4E     .7     LDA #'N
941C- 20 B9 94         JSR SYNSTR
941F- A9 44            LDA #'D
9421- 20 B9 94         JSR SYNSTR
9424- A9 01            LDA #1           SET AND OPERATOR
9426- 8D DD 94         STA ANDOR
9429- D0 C3            BNE .4           ...ALWAYS
                *---OR OPERATOR------------------
942B- A9 52     .8     LDA #'R
942D- 20 B9 94         JSR SYNSTR
9430- A9 00            LDA #0           SET OR OPERATOR
9432- 8D DD 94         STA ANDOR
9435- F0 B7            BEQ .4           ...ALWAYS
                *---NOT OPERATOR-----------------
9437- A9 4F     .9     LDA #'O
9439- 20 B9 94         JSR SYNSTR
943C- A9 54            LDA #'T
943E- 20 B9 94         JSR SYNSTR
9441- A9 01            LDA #1           SET NOT OPERATOR "TRUE"
9443- 8D DC 94         STA NOT
9446- D0 A6            BNE .4           ...ALWAYS
```

```
                 2920 *---FETCH OPERAND---------------
9448- 8D E4 94   2930 .10    STA ROPR
944B- A9 27      2940        LDA #$27          CHECK FOR APOSTROPHE
944D- 20 B9 94   2950        JSR SYNSTR
9450- AC E3 94   2960        LDY RINDEX
9453- B1 1B      2970        LDA (RSTR),Y      GET OPERAND
9455- 8D E5 94   2980        STA ROPD
9458- EE E3 94   2990        INC RINDEX
945B- A9 27      3000        LDA #$27          ANOTHER APOSTROPHE
945D- 20 B9 94   3010        JSR SYNSTR
                 3020 *---EVALUATE RELATIONAL OPERATION
9460- AD DF 94   3030        LDA NEW
9463- 8D E0 94   3040        STA LAST          LAST = NEW
9466- A9 00      3050        LDA #0            NEW = FAIL
9468- 8D DF 94   3060        STA NEW
946B- AC E4 94   3070        LDY ROPR          OPERATOR
946E- AD E1 94   3080        LDA KEY           LATEST KEY
9471- CD E5 94   3090        CMP ROPD          COMPARE TO OPERAND
9474- F0 08      3100        BEQ .11           THEY ARE EQUAL
9476- 90 0C      3110        BCC .12           KEY < OPERAND
9478- C0 3E      3120        CPY #'>           KEY > OPERAND
947A- F0 0C      3130        BEQ .13           SUCCESS!
947C- D0 0F      3140        BNE .14           FAIL.
947E- C0 3D      3150 .11    CPY #'=
9480- F0 06      3160        BEQ .13           SUCCESS
9482- D0 09      3170        BNE .14           FAIL
9484- C0 3C      3180 .12    CPY #'<
9486- D0 05      3190        BNE .14           FAIL
9488- A9 01      3200 .13    LDA #1            FLAG SUCCESS
948A- 8D DF 94   3210        STA NEW
                 3220 *---PERFORM NOT OPERATION--------
948D- AD DC 94   3230 .14    LDA NOT           IF NOT, TOGGLE NEW
9490- F0 0D      3240        BEQ .17           NOT NOT
9492- AD DF 94   3250        LDA NEW
9495- 49 01      3260        EOR #1
9497- 8D DF 94   3270        STA NEW
949A- A9 00      3280        LDA #0            CLEAR NOT
949C- 8D DC 94   3290        STA NOT
                 3300 *---PERFORM AND/OR OPERATION-----
949F- AD E0 94   3310 .17    LDA LAST
94A2- AC DD 94   3320        LDY ANDOR
94A5- F0 09      3330        BEQ .18           OR
94A7- 2D DF 94   3340        AND NEW           AND
94AA- 8D DF 94   3350        STA NEW
94AD- 4C EE 93   3360        JMP .4
94B0- 0D DF 94   3370 .18    ORA NEW
94B3- 8D DF 94   3380        STA NEW
94B6- 4C EE 93   3390        JMP .4
                 3400 *-------------------------------
94B9- 8D DB 94   3410 SYNSTR STA HOLD      SAVE CHAR
94BC- AC E3 94   3420 .1     LDY RINDEX
94BF- B1 1B      3430        LDA (RSTR),Y
94C1- EE E3 94   3440        INC RINDEX
94C4- C9 20      3450        CMP #'            IGNORE BLANKS
94C6- F0 F4      3460        BEQ .1
94C8- CD DB 94   3470        CMP HOLD
94CB- F0 03      3480        BEQ .2
94CD- 4C C9 DE   3490        JMP AS.SYNERR
94D0- 60         3500 .2     RTS
                 3510 *-------------------------------
                 3520 PRINT.FIELD
94D1- AC DE 94   3530        LDY FL
94D4- 20 ED FD   3540 .1     JSR MON.COUT
94D7- 88         3550        DEY
94D8- D0 FA      3560        BNE .1
94DA- 60         3570        RTS
                 3580 *-------------------------------
94DB-            3590 HOLD   .BS 1
94DC-            3600 NOT    .BS 1
94DD-            3610 ANDOR  .BS 1
94DE-            3620 FL     .BS 1
94DF-            3630 NEW    .BS 1
94E0-            3640 LAST   .BS 1
94E1-            3650 KEY    .BS 1
94E2-            3660 BINDEX .BS 1
94E3-            3670 RINDEX .BS 1
94E4-            3680 ROPR   .BS 1
94E5-            3690 ROPD   .BS 1
                 3700 *-------------------------------
```

&Amper-Magic T.M.

MACHINE LANGUAGE SPEED WHERE IT COUNTS...

IN YOUR PROGRAM!

Some routines on this disk are:

Binary file info
Delete array
Disassemble memory
Dump variables
Find substring
Get 2-byte values
Gosub to variable
Goto to variable
Hex memory dump
Input anything
Move memory
Multiple poke decimal
Multiple poke hex
Print w/o word break
Restore special data
Speed up Applesoft
Speed restore
Store 2-byte values
Swap variables

For the first time, Amper-Magic makes it easy for people who don't know machine language to use its power! Now you can attach slick, finished machine language routines to your Applesoft programs in seconds! And interface them by name, not by address!

You simply give each routine a name of your choice, perform the append procedure once at about 15 seconds per routine, and the machine language becomes a permanent part of your BASIC program. (Of course, you can remove it if you want to.)

Up to 255 relocatable machine language routines can be attached to a BASIC program and then called by name. We supply some 20 routines on this disk. More can be entered from magazines. And more library disks are in the works.

These routines and more can be attached and accessed easily. For example, to allow the typing of commas and colons in a response (not normally allowed in Applesoft), you just attach the Input Anything routine and put this line in your program:

```
xxx   PRINT "PLEASE ENTER THE DATE. "; : & INPUT,DATE$
```

&-MAGIC makes it Easy to be Fast & Flexible!

PRICE: $75

Anthro - Digital Software
P.O. Box 1385
Pittsfield, MA 01202

. The People - Computers Connection

&-Magic and Amper-Magic are trademarks of Anthro-Digital, Inc.
Applesoft is a trademark of Apple Computer, Inc.

Star-tling Stunts....................Bill Morgan & Mike Laumer

In most assemblers, including the S-C Macro Assembler, you can use the character "*" in the operand of an instruction to mean the current value of the location counter. (The location counter is a variable used by the assembler to keep track of where the next byte of object code goes.) Here are a couple of simple examples of using the *, from page 6-2 of the Macro Assembler manual:

```
0800- 03         1000 QT      .DA #QTSZ
0801- 41 42 43   1010         .AS /ABC/
0003-            1020 QTSZ    .EQ *-QT-1
                 1030
0804- 00 00      1040 VAR     .DA *-*
                 1050
0806-            1060 FILLER .BS $900-*
0900-            1070 END     .EQ *
```

The QT, QTSZ example uses the * to help calculate the length of a string of characters. The VAR line uses "*-*" to define a variable as having a value of zero.

The expression labelled FILLER causes the assembler to skip ahead to $900. This has much the same effect as .OR $900, but it won't cause the assembler to close a target file, the way .OR would.

One thing Bill wanted was an expression to have the assembly skip up to the beginning of the next page, no matter what that page might be. Here's what we came up with:

```
0800- 34 12      1000 START  .DA $1234
0802-            1010 FILL   .BS *+255/256*256-*
0900- 45 23      1020 END    .DA $2345
```

If you change the origin to $C00, END will move to $D00. With this coding, END will always be $100 above START. Note that there is no precedence when the assembler is evaluating an expression. Terms are taken strictly left-to-right. But notice how smart the expression cracker in the assembler is! It knows that a "*" between numbers or labels means "multiply", and a "*" between arithmetic operators means "location counter".

In the American Heart Association CPR project Mike uses lots of overlays, and has to make sure that modules don't grow above a certain address. He does it by putting lines like these at the end of a module:

```
1000          .DO *>LIMIT
1010    !!! PROGRAM TOO BIG !!!
1020          .FIN
```

Here's an example, to keep a program below the Hi-res pages:

```
          1000         .OR $1FFE
          1010         .DA $4321
          1020         .DO *>$2000
          1030   !!! PROGRAM TOO BIG !!!
          1040         .FIN
```

That will assemble just fine:

```
                         1000         .OR $1FFE
          1FFE- 21 43    1010         .DA $4321
                         1020         .DO *>$2000
                         1040         .FIN

          0000 ERRORS IN ASSEMBLY
```

But, try inserting another line:

```
          1015         .DA $1234
```

Here's what happens:

```
          *** BAD OPCODE ERROR
           1030   !!! PROGRAM TOO BIG !!!

          0001 ERRORS IN ASSEMBLY
```

The key to this technique is putting a couple of blanks at the beginning of line 1030. That way, the assembler tries to parse "!!!" as an opcode, and reports an error during pass one, before any code has been generated.

You should be very careful about using "*", and experiment on a test disk when trying something new. For example, take another look at line 1060 in the first listing. If you put "*-$900" for the operand, that would be negative. The result would be $FF07, which would try to write 65,287 zero bytes onto your target file. The next thing you see is probably DISK FULL!

That's about all the tricky things we have room for right now. We hope these hints will help you to navigate "by the stars" in your programming. Just remember to experiment carefully with the * operand before using it in vital programs. There are also many pitfalls on this road!

Promising New Book

I just received an advance copy of a forthcoming book by Jules Gilder (a long-time AAL subscriber), titled "Now That You Know Apple Assembly Language, What Can You Do With It?" As the title implies, this will be an intermediate level look at really using assembly language in your Apple. It looks good. As soon as I have details about price and publication date, I'll let you know.

A Sometimes Useful Patch..................Bob Sander-Cederlof

Sometimes you would like to see all the hex bytes a macro produces, but not the expanded lines of source code. The >LIST MOFF directive turns off both, but with the following three byte patch you can see the hex bytes for each macro call.

```
Motherboard version:     :$218B:0    (was 03)
                         :$21B3:0    (was 05)
                         :$21E2:0    (was 10)

RAM Card version:        :$C083 C083    (enable writing)
                         :$E2D7:0    (was 03)
                         :$E2FF:0    (was 05)
                         :$E32E:0    (was 10)
```

Don't make these into permanent patches, because there will be times when you want to use the .LIST directives normally. If you feel like making the changes often, you might make two separate versions of the assembler, or make some EXEC files to do the patching on demand.

RAM/ROM PROGRAM DEVELOPMENT BOARD $35.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip. Use a 6116 type RAM chip for program development or just extra memory. Plugin a programmed 2716 EPROM to keep your favorite routines 'on-line'. Maps into $Cn00-$CnFF and $C800-$CFFF memory space. Instructions & circuit diagram provided.

The 'MIRROR': Firmware for Apple-Cat $29.00

Communications ROM plugs directly into Novation's modem card. Three basic modes: Dumb Terminal, Remote Console & Programmable Modem. Added features include: Printer buffer, Pulse or Tone dialing, true dialtone detection, audible ring detect and ring-back option. Directly supports many 80-column boards (even while printing) and Apple's Comm card commands. (Apple-Cat Hardware differences prevent 100% interchangability with Comm card.) Includes Hayes-to-AppleCat register equivalances for software conversion. Telephone Software Connection (213-516-9430) has several programs which support the 'MIRROR'.

The 'PERFORMER': Smarts For Your Printer $49.00

Get the most from your smart printer by adding intelligence to your 'dumb' interface card. The PERFORMER Board plugs into any Apple slot for immediate access (no programs to find and load). Easily select printer fonts and many other features via a user-friendly menu. Replaces manual printer set-up. No need to remember ESC commands. Also provides TEXT and GRAPHICS screen dumps. Compatible with Apple, Tymac, Epson, Microtek and similar 'dumb' Centronics type parallel I/F boards. Specify printer: EPSON MX80 W/Graftrax-80, EPSON MX100, EPSON MX80/MX100 W/Graftrax Plus, NEC 8023A, C.Itoh 8510 (ProWriter), OKI Microline 82A/83A W/OKIGRAPH. (OKI Bonus: The PERFORMER Generates ENHANCED and DOUBLE STRIKE Fonts)

Avoid A $3.00 Shipping/Handling Charge By Mailing Full Payment With Order

RAK-WARE
41 Ralph Road
West Orange NJ 07052

***** SAY YOU SAW IT IN 'APPLE ASSEMBLY LINE'! *****

Source Code for a Word Processor..........Bob Sander-Cederlof

I finally have had to face it. I am never going to have time to finish the S-C word processor. It is certainly usable, because we have been using it here for months now. And we use it a lot, writing the newsletter, manuals, letters, etc. My father-in-law uses it, and so does my best friend, Fred. Fred's 11-year-old daughter is also using it, and loves it. She is currently typing a research paper using it.

I know it is easy to use, because I didn't even give Fred a list of commands, let alone a reference manual. Of course, I did sit down with them for a few hours at the first, because they had never even seen a word processor before.

In power, it is somewhere between Applewriter 1.1 and Applewriter II. It is similar in operation to Applewriter 1.1, and works in 40-column mode only. It requires a lower-case display and shift-key mod.

It can read Applewriter 1.1 files, and instantly convert them to standard ASCII form. Normally it uses standard Apple text files (type T in the catalog). Of course, with Bobby Deen's help, I built in FAST read and write of those text files. Faster than binary files, actually. Something like 100 sectors in 7 seconds, if I remember correctly.

I want to make a deal with you. I'll send you the complete commented source code on disk, together with a few sample text files. The text files will describe the command repertoire. If you are already familiar with Applewriter 1.1, you won't have any trouble at all. The assembled word processor will also be there, in case you don't have the S-C Macro Assembler.

But if you do have my assembler, you can proceed to modify, improve, augment, enhance, and so on, to your heart's content.

I'll send you the disk, if you'll send me $50. Or your charge card numbers, of course. I also want your commitment to keep this in the family. You know, don't go out and write a manual and wrap it in a fancy cover and call it YOUR product!

If you do enhance it, send in your additions and we'll make this a joint effort. With all of us working on it, we may soon have the world's best word machine!

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)